

Discrete State Transition Algorithm for Unconstrained Integer Optimization Problems

Xiaojun Zhou · David Yang Gao ·
Chunhua Yang

Received: date / Accepted: date

Abstract we focus on a recently new intelligent optimization algorithm called discrete state transition algorithm, for solving integer optimization problems. Firstly, we summarize some key elements for discrete state transition algorithm to guide its well development. Several intelligent operators are designed for local exploitation and global exploration. Then, a dynamic adjustment strategy “risk and restore in probability” is proposed to gain global convergence with high probability. Finally, numerical experiments are carried out to test the effectiveness of the proposed algorithm, and they show that the similar intelligent operators can be applied to ranging from traveling salesman problem, boolean integer programming, to discrete value selection problem, which indicates the robustness and adaptability of the proposed intelligent elements.

Keywords state transition algorithm · integer programming · traveling salesman problem · maximum cut problem · discrete value selection

Xiaojun Zhou

School of Information Science and Engineering, Central South University, Changsha 410083, China.

School of Science, Information Technology and Engineering, University of Ballarat, Victoria 3353, Australia.

David Yang Gao

School of Science, Information Technology and Engineering, University of Ballarat, Victoria 3353, Australia.

Chunhua Yang

School of Information Science and Engineering, Central South University, Changsha 410083, China.

1 Introduction

In this paper, we consider the following unconstrained integer optimization problem

$$\min f(x), \quad (1)$$

where, $x = (x_1, \dots, x_n)$, $x_i \in \mathcal{I} \subset \mathcal{Z}^m$, $i = 1, \dots, n$.

Generally speaking, the above optimization problem is NP-hard, which can not be solved in polynomial time (by enumeration, there exist m^n choices). A direct method is to adopt the so called “divide-and-conquer” strategy, which separates the optimization problem into several subproblems and then solve these subproblems step by step. Branch and bound (B&B), branch and cut (B&C), and branch and price (B&P) belong to this kind; however, these methods are essentially in exponential time. An indirect method is to relax the optimization problem by loosening its integrality constraints to continuity and then solve the continuous relaxation problem or its Lagrangian dual problem, including LP-based relaxation, SDP-based relaxation, Lagrangian relaxation, etc. Nevertheless, when rounding off the relaxation solution, there may cause some infeasibility or can only get approximate solution, and when using Lagrangian dual, there may exist duality gap between the primal and the dual problem [3, 4, 5, 8].

On the other hand, some stochastic algorithms, such as genetic algorithm (GA) [1, 15], simulated annealing (SA) [6, 17], ant colony optimization (ACO) [2, 11], are also widely used for combinatorial optimization problems, which aim to obtain “good solutions” in reasonable time. In terms of the concepts of state and state transition, a new heuristic search algorithm called state transition algorithm (STA) has been proposed recently, which exhibits fantastic results in continuous function optimization [18, 19, 20]. In [14], three intelligent operators (geometrical operators) named swap, shift and symmetry have been designed for discrete STA to solve the traveling salesman problem (TSP), and it shows that the discrete STA outperforms its competitors in respect to both time complexity and search ability. To better develop discrete STA for medium-size or large-size discrete optimization problems, in the study, we firstly build the framework of discrete state transition algorithm and propose the five key elements for discrete STA, of which, the representation of a decision variable in discrete STA, the local and global operators and the dynamic adjustment strategy are mainly studied. Four geometrical operators named swap, shift, symmetry and substitute are designed, which are intelligent due to their adaptability in various types of integer programming. The mixed strategies of “greedy criterion” and “risk and restore in probability” are proposed, in which, “greedy criterion” and “restore in probability” are used to guarantee good convergence performance, and “risk a bad solution in probability” aims to escape from local optimality. Some applications ranging from traveling salesman problem to discrete value selection problem are studied.

2 The framework of discrete state transition algorithm

If a solution to a specific optimization problem is described as a state, then the transformation to update the solution becomes a state transition. Without loss of generality, the unified form of discrete state transition algorithm can be described as

$$\begin{cases} x_{k+1} = A_k(x_k) \oplus B_k(u_k) \\ y_{k+1} = f(x_{k+1}) \end{cases}, \quad (2)$$

where, $x_k \in \mathcal{Z}^n$ stands for a current state, corresponding to a solution of a specific optimization problem; u_k is a function of x_k and historical states; $A_k(\cdot)$, $B_k(\cdot)$ are transformation operators, which are usually state transition matrixes; \oplus is a operation, which is admissible to operate on two states; f is the cost function or evaluation function.

The first element we want to mention is the representation of a solution to a discrete optimization problem. In discrete STA, we choose special representations, which can be easily manipulated by the intelligent operators. Why we call the operators “intelligent” due to its geometrical property (swap, shift, symmetry and substitute), and a intelligent operator has the same geometrical function for different representations. The big advantage of such representations and operators is that, after each state transformation, the newly created state is always feasible, avoiding the trouble in dealing with constraints.

The second important thing in state transition algorithm is that, when a transformation operator is exerted on current state, the next state is not deterministic, that is to say, there are possibly different choices for the next state. It is not difficult to imagine that all possible choices will constitute a set, or a neighborhood. Then we execute several times of transformation (search enforcement *SE*) on current state, to sampling in the neighborhood. Sampling is the first key element in state transition algorithm, which can reduce the search space and avoid enumeration.

As a intelligent optimization algorithm, the discrete state transition algorithm also comprises the following key elements:

(1) local exploitation and global exploration. In optimization algorithms, it is very significant to design good local and global operators. The local exploitation can guarantee high precision of a solution and convergent performance of a algorithm, and the global exploration can avoid getting trapped into local minima or prevent premature convergence. In discrete optimization, it is extremely difficult to define a “good” local optimal solution due to its dependence on a problem’s structure, which leads to the same difficulty in the definition of local exploitation and global exploration. Anyway, in the discrete state transition algorithm, we define the slow change to current solution by a transformation as local exploitation, while the big change to current solution by a transformation as global exploration.

(2) self learning and regular communication. State transition algorithm behaves in two forms, one is individual-based, the other is population-based, which is certainly a extended version. The individual-based state transition

algorithm focuses on self learning, in other words, with emphasis on the operators' designing and dynamic adjustment (details given in the following). Undoubtedly, communication among different states is a promising strategy for state transition algorithm, as indicated in [20]. Through communication, states can share information and cooperate with each other. However, how to communicate and when to communicate are key issues. In continuous state transition algorithm, intermittent exchange strategy was proposed, which means that states communicate with each other at a certain frequency in a regular way.

(3) dynamic adjustment. It is a potentially useful strategy for state transition algorithm. In the iteration process of a algorithm, the fitness value can decrease sharply in the early stage, but it stagnates in the late stage, due to the static environment. As a result, some perturbation should be added to activate the environment. In fact, dynamic adjustment can be understood and implemented in various ways. For example, the alternative use of different local and global operators is dynamic adjustment to some extent. Then, we can change the search enforcement, vary the cost function, reduce the dimension, etc. Of course, "risk a bad solution in probability" is another dynamic adjustment, which is widely used in simulated annealing (SA). In SA, the Metropolis criterion [9] is used to accept a bad solution:

$$\text{probability } p = \exp\left(\frac{-\Delta E}{k_B T}\right), \quad (3)$$

where, $\Delta E = f(x_{k+1}) - f(x_k)$, k_B is the Boltzmann probability factor, T is the temperature to regulate the process of annealing. In the early stage, temperature is high, and it has big probability to accept a bad solution, while in the late stage, temperature is low, and it has very small probability to accept a bad solution, which is the key point to guarantee the convergence. We can see that the Metropolis criterion has the ability to escape from local optimality, but on the other hand, it will miss some "good solutions" as well.

In this study, we focus on the individual-based STA, and the main process of discrete STA is shown in the pseudocode as follows

```

1: repeat
2:   [Best,fBest]  $\leftarrow$  swap(funcn,Best,fBest,SE,n,m_a)
3:   [Best,fBest]  $\leftarrow$  shift(funcn,Best,fBest,SE,n,m_b)
4:   [Best,fBest]  $\leftarrow$  symmetry(funcn,Best,fBest,SE,n,m_c)
5:   [Best,fBest]  $\leftarrow$  substitute(funcn,Best,fBest,SE,set,n,m_d)
6:   if fBest < fBest* then ▷ greedy criterion
7:     Best*  $\leftarrow$  Best
8:     fBest*  $\leftarrow$  fBest
9:   end if
10:  if rand <  $p_1$  then ▷ restore in probability, only in DSTA
11:    Best  $\leftarrow$  Best*
12:    fBest  $\leftarrow$  fBest*
13:  end if
14: until the maximum number of iterations is met

```

As for detailed explanations, swap function in above pseudocode is given as follows for example

```

1: State  $\leftarrow$  op_swap(Best, SE, n, m_a)
2: [newBest, fGBest]  $\leftarrow$  fitness(funfcn, State)
3: if fGBest < fBest then                                      $\triangleright$  greedy criterion
4:   Best  $\leftarrow$  newBest
5:   fBest  $\leftarrow$  fGBest
6: else
7:   if rand <  $p_2$  then                                        $\triangleright$  risk in probability, only in DSTA
8:     Best  $\leftarrow$  newBest
9:     fBest  $\leftarrow$  fGBest
10:  end if
11: end if

```

From the pseudocodes, we can find that in the simple STA, only “greedy criterion” is adopted to accept a new “Best*”, while in dynamic STA, in the whole, “greedy criterion” is adopted to keep the incumbent “Best*”, in the partial, a bad solution “Best” is accepted in each state transformation at a probability p_2 , and in the same while, the “Best*” is restored in the process at another probability p_1 . The “risk a bad solution in probability” strategy aims to escape from local optimal, while the “greedy criterion” and “restore the incumbent best solution in probability” are to guarantee a good convergence.

3 Theoretical analysis of the discrete STA

Firstly, we define the concept of convergence for discrete STA

$$|f(x_k) - f(x^*)| \leq \epsilon, \forall k > N, \quad (4)$$

while, x^* is a global optimum of a discrete optimization problem, ϵ is a small constant, and N is a natural number. If $\epsilon > 0$, we can say that the algorithm converges to a ϵ - *suboptimal solution*; if $\epsilon = 0$, we can say that the algorithm converges to a *global minimum*. However, if x^* is a local minimum solution, then we can say that the algorithm converges to a ϵ - *suboptimal solution* and a *local minimum* for $\epsilon > 0$ and $\epsilon = 0$, respectively.

Theorem 1 the discrete STA can at least converge to a *local minimum*.

Proof. Let suppose the maximum number of iterations (denoted by M) is big enough, considering that the “greedy criterion” is used to keep the incumbent best, then there must exist a number $N < M$, when $k > N$, no update of better solution will happen. That is to say, $f(x_k) = f(x_N), \forall k > N$, where x_N is the solution in the N th iteration. Denote x_N as the local minimum solution x^* , and then $|f(x_k) - f(x^*)| = 0$.

Theorem 2 the discrete STA can converge to a *global minimum* in probability.

Proof. Let suppose $x^* = (a_1, \dots, a_n)$ is a global minimum solution, and

$x_N = (b_1, \dots, b_n)$ is the N th best solution. If $x_N = x^*$, according to the “greedy criterion”, $f(x_k) = f(x_N), \forall k > N$, which means that it converges to x^* . Otherwise, there must exist a transformation, either swap, shift, symmetry or substitute, such that $x_{N+l_1} = (a_1, \dots, b_n)$, which means that after l_1 iterations, b_1 will be changed into a_1 . If $f(x_{N+l_1}) < f(x_N)$, the x_{N+l_1} is kept as incumbent best for next iteration, else, x_{N+l_1} is also kept as incumbent best for next iteration in probability. Following the similar way, after at most $l_2 + \dots + l_n$ iterations, $x_{N+l_1+\dots+l_n}$ will be (a_1, \dots, a_n) in probability.

From the proof of Theorem 2, we can find that dynamic STA has higher probability than the simple STA to gain the global optimum. Some applications are given to describe the details in the following.

4 Application for traveling salesman problem

Suppose $\mathcal{N} = \{1, \dots, n\}$ is the set of cities, the traveling salesman problem can be described as: given a set of n cities and the distance d_{ij} for each pair of cities i and j , find a roundtrip of minimal total length visiting each city exactly once. Typically, the traveling salesman problem is usually modeled as the following two representations [12].

(LP-TSP):

$$\begin{aligned}
 & \min_{x_{ij}} \quad \sum_{i=1}^n \sum_{j=1}^n x_{ij} d_{ij} \\
 & s.t. \quad \sum_{i=1}^n x_{ij} = 1, \forall j \in \mathcal{N} \\
 & \quad \sum_{j=1}^n x_{ij} = 1, \forall i \in \mathcal{N} \\
 & \quad \sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geq 1, \\
 & \quad \forall S \subset \mathcal{N}, \bar{S} \subset \mathcal{N} \setminus S \\
 & \quad x_{ij} \in \{0, 1\}, \forall i, j \in \mathcal{N},
 \end{aligned} \tag{5}$$

where, the decision variable x_{ij} is defined by

$$x_{ij} = \begin{cases} 1, & \text{if city } i \text{ is followed by city } j \\ 0, & \text{otherwise} \end{cases} \tag{6a}$$

$$\tag{6b}$$

(QP-TSP):

$$\begin{aligned}
 & \min_{x_{ij}} \quad \sum_{i=1}^n \sum_{k=1}^n \sum_{j=1}^n x_{ij} d_{ik} (x_{k(j+1)} + x_{k(j-1)}) \\
 & s.t. \quad \sum_{i=1}^n x_{ij} = 1, \forall j \in \mathcal{N} \\
 & \quad \sum_{j=1}^n x_{ij} = 1, \forall i \in \mathcal{N} \\
 & \quad x_{ij} \in \{0, 1\}, \forall i, j \in \mathcal{N},
 \end{aligned} \tag{7}$$

here, the decision variable x_{ij} is defined by

$$x_{ij} = \begin{cases} 1, & \text{if city } i \text{ is in the } j\text{th position} \\ 0, & \text{otherwise} \end{cases} \quad (8a)$$

$$(8b)$$

We can find that the model of linear programming (LP) based TSP is different from the model of quadratic programming (QP) based TSP in two aspects, one is the definition of the decision variable x_{ij} , the other is that (LP-TSP) has one more constraint than (QP-TSP). Taking the difficulty of dealing with constraints into consideration, in discrete STA, we use another simple representation which can be easily manipulated by intelligent operators.

4.1 Transformation operators for TSP

As for a n -city traveling salesman problem, a permutation of $\{1, 2, \dots, n\}$ is used to represent a solution to the problem in discrete STA. Based on the representation, three special transformation operators are proposed to illustrate local and global search.

(1) swap transformation

$$x_{k+1} = A_k^{swap}(m_a)x_k, \quad (9)$$

where, $A_k^{swap} \in \mathbb{R}^{n \times n}$ is called swap transformation matrix, m_a is a constant integer called swap factor to control the maximum number of positions to be exchanged, while the positions are random. If $m_a = 2$, we call the swap operator local exploitation, and if $m_a \geq 3$, the swap operator is regarded as global exploration in this case. Figure 1 gives the function of the swap transformation graphically when $m_a = 2$.

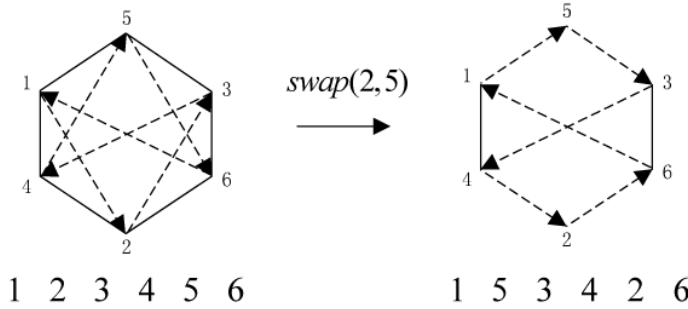


Fig. 1 illustration of swap transformation for TSP

(2) shift transformation

$$x_{k+1} = A_k^{shift}(m_b)x_k, \quad (10)$$

where, $A_k^{shift} \in \mathbb{R}^{n \times n}$ is called shift transformation matrix, m_b is a constant integer called shift factor to control the maximum length of consecutive positions to be shifted. By the way, the selected position to be shifted after and positions to be shifted are chosen randomly. Similarly, shift transformation is called local exploitation and global exploration when $m_b = 1$ and $m_b \geq 2$ respectively. To make it more clearly, if $m_b = 1$, we set position 3 to be shifted after position 5, as described in Figure 2.

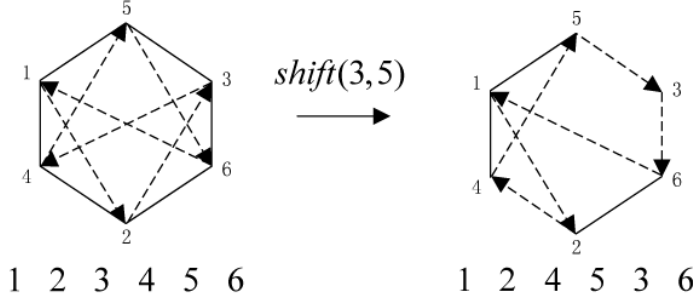


Fig. 2 illustration of shift transformation for TSP

(3) symmetry transformation

$$x_{k+1} = A_k^{sym}(m_c)x_k, \quad (11)$$

where, $A_k^{sym} \in \mathbb{R}^{n \times n}$ is called symmetry transformation matrix, m_c is a constant integer called symmetry factor to control the maximum length of subsequent positions as center. By the way, the component before the subsequent positions and consecutive positions to be symmetrized are both created randomly. Considering that the symmetry transformation can make big change to current solution, it is intrinsically called global exploration. For instance, if $m_c = 0$, let choose component 3, then the subsequent position or the center is $\{\emptyset\}$, the consecutive positions are $\{4, 5\}$, and the function of symmetry transformation is given in Figure 3.

4.2 TSP instances for test

To evaluate the performance of the simple STA and the dynamic STA (DSTA), 4 medium-size TSP instances are used for test. We set $m_a = 2, m_b = 1, m_c = 0, m_d = 1, p_1 = 0.1459, p_2 = 0.0557$ and the maximum number of iterations at 1500. Programs are run independently for 20 trails for each algorithms in MATLAB R2010b (version of 7.11.0.584) on Intel(R) Core(TM) i3-2310M CPU @2.10GHz under Window 7 environment, and for each run, it costs about

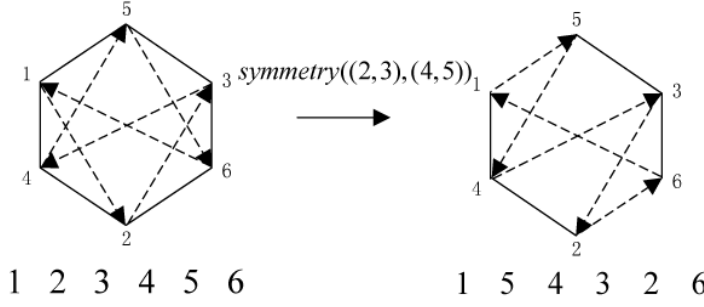


Fig. 3 illustration of symmetry transformation for TSP

0.5 minutes. Some statistics as well as the “error” are computed for comparison, where, “error” is defined by

$$error = \frac{best - optimum}{optimum} \times 100\%,$$

where, *best* is the best result achieved by discrete STA, *optimum* is the incumbent best result in TSPLIB [10]. Experimental results for these TSP instances are given in Table 1.

Table 1 experimental results for TSP instances

<i>instance</i>	<i>optimum</i>	<i>algorithm</i>	<i>best</i>	<i>mean</i>	<i>s.t.</i>	<i>error</i>
gr96.tsp	512.3094	STA	525.0124	544.9243	13.2005	2.48%
		DSTA	515.5143	549.9838	17.1709	0.63 %
kroA100.tsp	2.1285e4	STA	2.2214e4	2.2970e4	515.9307	4.37%
		DSTA	2.1910e4	2.2740e4	700.3377	2.94%
kroC100.tsp	2.0751e4	STA	2.1570e4	2.2525e4	679.3821	3.95%
		DSTA	2.1084e4	2.2186e4	703.0488	1.60%
gr120.tsp	1.6665e3	STA	1.6826e3	1.7269e3	30.0698	0.97%
		DSTA	1.6418e3	1.7257e3	34.4110	-1.48%

As can be seen from Table 1, the biggest error is no more than 5%, and the DSTA gets much more better results than the STA. Especially for the gr120.tsp, as emphasized by the bold font, DSTA achieves the better result than the incumbent best result given in the TSPLIB, with the sequence of (82 11 51 23 9 103 119 4 38 7 56 41 87 14 75 44 46 20 50 98 42 17 118 49 13 113 69 65 68 91 58 79 100 33 52 25 108 43 107 18 19 117 31 66 85 22 81 94 86 78 15 59 76 1 16 61 29 120 32 30 92 28 45 74 105 72 40 60 24 111 96 90 54 8 70 116 34 26 71 47 55 6 89 112 110 48 102 101 114 106 104 36 84 35 10 99 62 37 67 83 39 57 73 63 77 95 12 97 88 21 115 2 93 109 64 53 5 27 80 3) and length of route at 1.6418e3, which is illustrated in Figure 4.

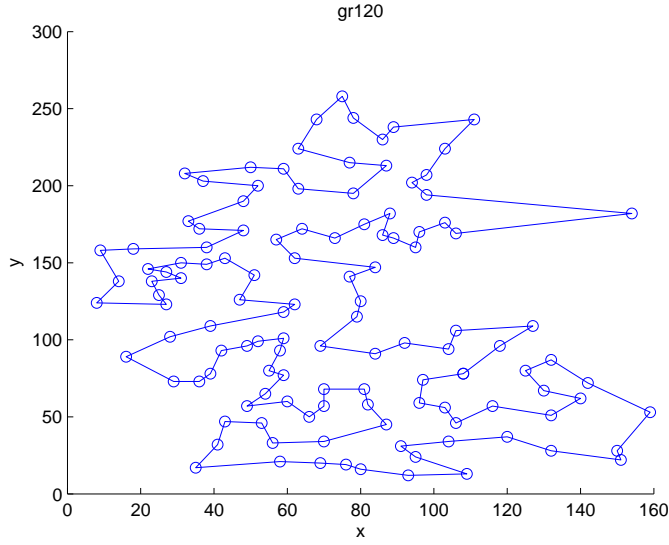


Fig. 4 the best route of gr120.tsp obtained by DSTA

5 Application for boolean integer programming

In boolean integer programming (BIP), a solution comprise a series of boolean values ($\mathcal{I} = \{0, 1\}$ or $\mathcal{I} = \{-1, 1\}$). Swap, shift and symmetry operators are also proposed for internal transformation (operators aiming to change the internal components of a sequence), and another operator called substitute is designed for external transformation (operator aiming to bring new components into a sequence). It should be noted that $\mathcal{I} = \{0, 1\}$ is the same to $\mathcal{I} = \{-1, 1\}$ under such circumstances, although there exists a linear transformation relationship between them in other studies.

5.1 Transformation operators for boolean integer programming

As we said previously, the same intelligent operator has the same geometrical property for different applications. It is not difficult to imagine that the swap, shift and symmetry operators for boolean integer programming have the same formulation as that of traveling salesman problem. Let $\mathcal{I} = \{0, 1\}$, the illustrations of internal transformation are given from Figure 5 to Figure 7.

Now, let introduce the external transformation.

(4) substitute transformation

$$x_{k+1} = A_k^{sub}(m_d)x_k, \quad (12)$$

where, $A_k^{sub} \in \mathbb{R}^{n \times n}$ is called substitute transformation matrix, m_d is a constant integer called substitute factor to control the maximum number of positions to be substituted. By the way, the positions are randomly created. If

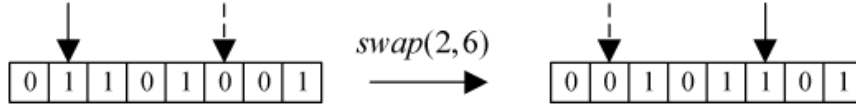


Fig. 5 illustration of swap transformation for BIP

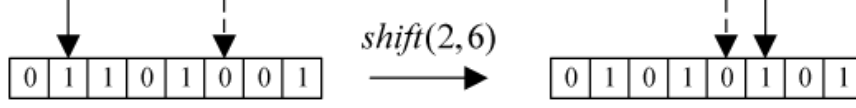


Fig. 6 illustration of shift transformation for BIP

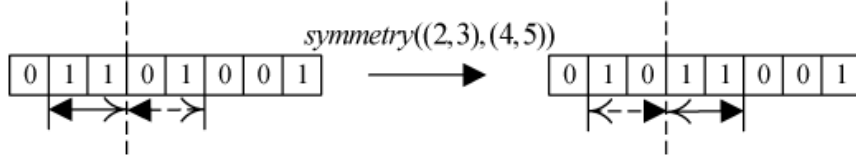


Fig. 7 illustration of symmetry transformation for BIP

$m_d = 1$, we call the substitute operator local exploitation, and if $m_d \geq 2$, the substitute operator is regarded as global exploration in this case. Figure 8 gives the function of the substitute transformation vividly when $m_d = 1$.

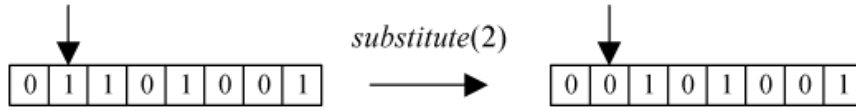


Fig. 8 illustration of substitute transformation for BIP

5.2 MAX-CUT instances for test

Let $G = (V, E)$ be an undirected graph with edge weight w_{ij} on $n + 1 = |V|$ vertices and $m = |E|$ edges, for each edge $(i, j) \in E$, the maximum cut problem (MAX-CUT) is to find a subset S of the vertex set V such that the total weight of the edges between S and its complementary subset $\bar{S} = V \setminus S$ is as large as possible.

(LP based MAX-CUT model)[7]:

Considering a variable y_{ij} for each edge $(i, j) \in E$, and assuming y_{ij} to be 1 if (i, j) is in the cut, and 0 otherwise, the MAX-CUT can be modeled as the

following linear programming (LP) optimization problem:

$$\begin{aligned} \max \quad & W(\mathbf{y}) = \sum_{i=1}^{n+1} \sum_{i < j, (i,j) \in E} w_{ij} y_{ij} \\ \text{s.t.} \quad & \mathbf{y} \text{ is the incidence vector of a cut.} \end{aligned} \quad (13)$$

Here the incidence vector $\mathbf{y} = \{y_{ij}\} \in \mathbb{R}^m$, where the m is the number of edges in the graph.

Let $\text{CUT}(G)$ denote the convex hull of the incidence vectors of cuts in G . Since maximizing a linear function over a set of points equals to maximizing it over the convex hull of this set of points, we can rewrite (13) to the following

$$\begin{aligned} \max \quad & W(\mathbf{y}) = \mathbf{c}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{y} \in \text{CUT}(G). \end{aligned} \quad (14)$$

where $\mathbf{c} = \{w_{ij}\} \in \mathbb{R}^m$.

(SDP based MAX-CUT model)[13]:

For a bipartition (S, \bar{S}) , with $y_i = 1$ if $i \in S$, and $y_i = -1$ otherwise, the MAX-CUT can also be formulated as the following semidefinite programming optimization problem:

$$\begin{aligned} \max \quad & W(\mathbf{y}) = \frac{1}{4} \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} w_{ij} (1 - y_i y_j) \\ \text{s.t.} \quad & \mathbf{y} \in \{-1, 1\}^{n+1}. \end{aligned} \quad (15)$$

Without loss of generality, if we fix the value of the last variable at 1, then the problem (15) is equivalent to the integer quadratic optimization problem

$$(\mathcal{P}) : \min \left\{ P(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} - \mathbf{x}^T \mathbf{c} : \mathbf{x} \in \{-1, 1\}^n \right\}, \quad (16)$$

where, $Q = \{Q_{ij}\}$ is a symmetric matrix with $Q_{ij} = w_{ij}$ ($i, j = 1, 2, \dots, n$), and $\mathbf{c} = -(w_{1(n+1)}, \dots, w_{n(n+1)})^T$. It is not difficult to find that a optimal solution \mathbf{x}^* to problem (16) corresponds to a optimal solution $(\mathbf{x}^*, 1)$ of original problem (15).

Considering that the SDP based MAX-CUT model is easier to manipulate for intelligent operators, it is adopted in discrete STA for simple representation. The parameters setting is the same to that in TSP instances. Under the circumstance, it takes about 8 seconds for each run. In the same way, we define the following “error”

$$\text{error} = \frac{\text{optimum} - \text{best}}{\text{optimum}} \times 100\%,$$

where, *best* is the best result achieved by discrete STA, *optimum* can be found in [13]. Experimental results for MAX-CUT instances are given in Table 2.

As can be seen from Table 2, the discrete STA has the ability to achieve the global minimum for all of the instances, and the difference between simple STA and DSTA is small in this case.

Table 2 experimental results for MAX-CUT instances

<i>instance</i>	<i>optimum</i>	<i>algorithm</i>	<i>best</i>	<i>mean</i>	<i>s.t.</i>	<i>error</i>
gr96	105328	STA	105328	105051	567	0
		DSTA	105328	104982	614	0
kroA100	5897392	STA	5897392	5864276	46302	0
		DSTA	5897392	5887930	29122	0
kroC100	5890760	STA	5890760	5860540	34277	0
		DSTA	5890760	5870613	31573	0
gr120	2156667	STA	2156667	2150378	7132	0
		DSTA	2156667	2150378	7132	0

6 Application for discrete value selection

Typically, the model of discrete value selection (DVS) problem is different from the model in (1), because the domain is defined as follows

$$x_i \in \mathcal{U} = \{u_1, \dots, u_m\}, u_j \in \mathbb{R}, j = 1, \dots, m. \quad (17)$$

By introducing a linear transformation

$$x_i = \sum_{j=1}^K u_j y_{ij}, \quad (18)$$

where

$$\sum_{j=1}^K y_{ij} = 1, y_{ij} \in \{0, 1\}, \quad (19)$$

then the discrete value selection can be rewritten to the equivalent constrained boolean integer programming problem [16].

In discrete STA, we only use the index of u_j to represent a solution, for example, a solution $(1, 3, 2)$ is corresponding to (u_1, u_3, u_2) , which is easy to be manipulated by the intelligent operators.

6.1 Transformation operators for discrete value selection

The intelligent operators swap, shift, symmetry and substitute for discrete value selection are similar to that in boolean programming problem. As a result, only illustrations of these transformations are given from Figure 9 to Figure 12.

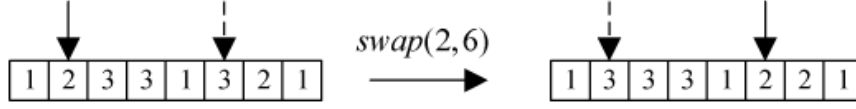


Fig. 9 illustration of swap transformation for DVS

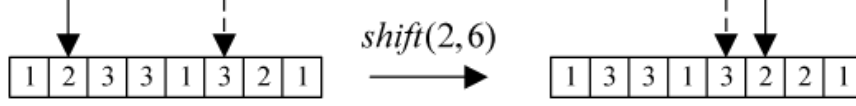


Fig. 10 illustration of shift transformation for DVS

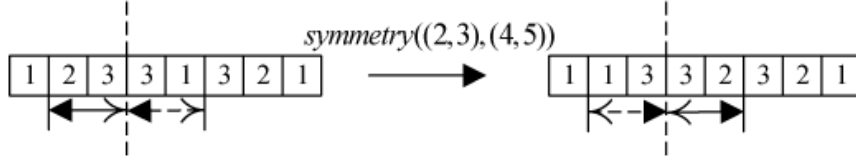


Fig. 11 illustration of symmetry transformation for DVS

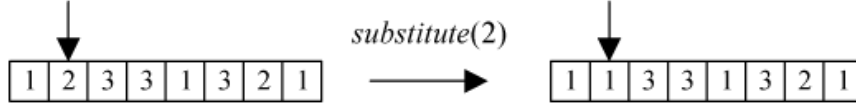


Fig. 12 illustration of substitute transformation for DVS

6.2 The integer Rosenbrock function for test

The continuous Rosenbrock function has been widely studied as a benchmark. Considering that the Rosenbrock function also has many integer local minima, in this paper, it is the first time to use it as a benchmark for integer programming problem. The integer Rosenbrock function is defined as

$$f_{rosenbrock} = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2], x_i \in \{-2, -1, 0, 1, 2\}.$$

It is not difficult to find that $x = (x_1, \dots, x_n)$ for any $x_i = 0, i = 1, \dots, n$ is a local minimum of the integer Rosenbrock function.

The parameters setting is the same to that of MAX-CUT instances except the maximum number of iterations, which is specified as 10, 20, 100, 200, 500, 2000 for $n = 5, 10, 20, 50, 100, 200$, respectively. Under the circumstance, 0.1, 0.15, 0.35, 0.6, 1.5, 6.0 seconds are consumed correspondingly. Experimental results for integer Rosenbrock function are given in Table 3.

As can be seen from Table 3, both the simple STA and the dynamic STA do well in the low dimensional cases. For large scale problem, the simple STA begins to have poor performance and even fails to find the global optimum.

Table 3 experimental results for integer Rosenbrock function

n	$optimum$	$algorithm$	$best$	$mean$	$s.t.$	$error$
5	0	STA	0	0	0	0
		DSTA	0	0	0	0
10	0	STA	0	0	0	0
		DSTA	0	0	0	0
20	0	STA	0	0	0	0
		DSTA	0	0	0	0
50	0	STA	0	0	0	0
		DSTA	0	0	0	0
100	0	STA	0	78.4500	46.0669	0
		DSTA	0	0	0	0
200	0	STA	107	155.4500	42.1495	-
		DSTA	0	0	0	0

On the other hand, the dynamic STA is more robust and adaptable, which verifies the Theorem 2 that dynamic STA has higher probability to gain global optimum than the simple STA.

7 Conclusion

In this paper, a new intelligent optimization algorithm named discrete state transition algorithm is studied for integer programming problem. It is the first time to build the framework for discrete state transition algorithm, and five key elements are discussed to better develop the algorithm. The representation of a feasible solution and the dynamic adjustment strategy are mainly studied. Various applications have shown the adaptability of the designed intelligent operators and experimental results have testified the effectiveness and efficiency of the proposed algorithm and strategies.

References

1. Ahmed, Z.H.: Genetic algorithm for the traveling salesman problem using sequential constructive crossoveroperator. International journal of biometrics & bioinformatics. 3(6), 96–105 (2010)
2. Dorigo, M., Gambardella, L.M.: Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE transaction on evolutionary computation. 1(1), 53–66 (1997)
3. Darby–Dowman, K., and Wilson, JM: Developments in linear and integer programming. Journal of the Operational Research Society. 53, 1065–1071 (2002)
4. Geoffrion, A.M., Marsten, R.E.: Integer programming algorithms: a framework and state-of-the-art survey. Management Science. 18(9), 456–491 (1972)
5. Jünger, M., Liebling, Th.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A.: 50 Years of Integer Programming 1958–2008. New York: Springer (2010)
6. Kirkpatrick, S., Gelatt C.D. and Vecchi, M.P.: Optimization by simulated annealing. Science. 220(4590), 671–680 (1983)

7. Krishnan, K., Mitchell, J.E.: A semidefinite programming based polyhedral cut and price approach for the maxcut problem. *Computational Optimization and Applications*. 35, 51–71 (2006)
8. Li, D., Sun, X.L.: *Nonlinear integer programming*. New York: Springer (2006)
9. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E.: Equation of state calculations by fast computing machines. *Journal of Chemical Physics*. 21(6), 1087–1092, (1953)
10. Reinelt, G.: TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*. 3(4), 376–384 (1991)
11. Schlüter, M., Egea, J.A., Banga, J.R.: Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Computers & operations research*. 36, 2217–2229 (2009)
12. Smith, K.: An argument for abandoning the traveling salesman problem as a neural-network benchmark. *IEEE Transaction on neural network*, 7(6), 1542–1544 (1996)
13. Wang, Z.B., Fang, S.C., Gao, D.Y., Xing, W.X.: Canonical dual approach to solving the maximum cut problem. online in *Journal of Global Optimization*. (2012)
14. Yang, C.H., Tang, X.L., Zhou, X.J., Gui, W.H.: State transition algorithm for traveling salesman problem. arXiv:1206.0329v1, to appear in the 31st Chinese Control Conference (2012)
15. Yokota, T., Gen, M., Li, Y.X.: Genetic algorithm for non-linear mixed integer programming problems and its applications. *Computers & industrial engineering*. 30(4), 905–917 (1996)
16. Yu, C.J., Teo, K.L., Bai, Y.Q.: An exact penalty function method for nonlinear mixed discrete programming problems. *Optimization letters*. (2011) doi:10.1007/s11590-011-0391-2
17. Zhang, D.F., Liu Y.K., Hallah, R.M., Leung, S.C.H.: A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European journal of operational research*. 203, 550–558 (2010)
18. Zhou, X.J., Yang, C.H., Gui, W.H.: Initial version of state transition algorithm. *International conference on digital manufacturing and automation (ICDMA)*. pp. 644–647 (2011)
19. Zhou, X.J., Yang, C.H., Gui, W.H.: A new transformation into state transition algorithm for finding the global minimum. *International conference on intelligent control and information processing (ICICIP)*. pp. 674–678 (2011)
20. Zhou, X.J., Yang, C.H., Gui, W.H.: State transition algorithm. arXiv:1205.6548v1, to appear in *Journal of Industrial and Management Optimization* (2012)